A Stateless Neighbour-Aware Cooperative Caching Protocol for Ad-Hoc Networks

Hugo Miranda Simone Leggio
Luís Rodrigues Kimmo Raatikainen

DI–FCUL TR–05–23

December 2005

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

A Stateless Neighbour-Aware Cooperative Caching Protocol for Ad-Hoc Networks*†

Hugo Miranda[‡] Simone Leggio[§] Luís Rodrigues[‡]
Kimmo Raatikainen[§]

[‡]Universidade de Lisboa
Departamento de Informática
{hmiranda,ler}@di.fc.ul.pt

[§]University of Helsinki
Computer Science Department
{simone.leggio,kimmo.raatikainen}@cs.helsinki.fi

December 2005

Abstract

Replication of data items among different nodes of a Mobile Ad-Hoc Network (MANET) is an efficient technique to increase data availability and improve access latency. This paper proposes a novel algorithm to distribute cached data items among nodes in a MANET. The algorithm combines a probabilistic approach with latency constraints such as the distance from both the source and the clients of the data item. In most scenarios, our approach allows any node to retrieve a data item from a nearby neighbour (often, just one hop away). The paper describes the algorithm and provides its performance evaluation for several different network configurations.

*This work was partially supported by the Middleware for Network Eccentric and Mobile Applications (MiNEMA) programme of the European Science Foundation

†Also Technical Report Number C-2005-76. Computer Science Department, University of Helsinki

1 Introduction

Information management in wireless ad-hoc networks is not a straightforward task. The inherently distributed nature of the ad-hoc environment, and the dynamic characteristics of both network topology and medium connectivity, are challenges for the efficient handling of data. There are several policies that can be followed when dealing with information management in ad-hoc networks. In particular, one must first choose whether to follow a centralised or decentralised approach.

Due to the nature of the ad-hoc environment, solutions that concentrate the data in a single entity must be discarded; the node hosting such a centralised entity would require significantly more resources than other nodes. Usually, ad-hoc networks are formed by peer nodes, with limited capabilities, so it is not practical to elect one node to act as a repository for the data needed by the other nodes. Moreover, this approach introduces a single point of failure; this is unacceptable given that node failures are an integral part of ad-hoc networks (failures may happen due to voluntary departures, crashes, or simply due to medium impairments). A decentralised approach is, therefore, strongly favoured.

In a decentralised approach, the data items are spread among all the nodes of the network. The data dissemination algorithm should balance the need to provide data replication (to cope with failures) with the need to avoid excessive data redundancy (as nodes may have limited storage capability). Furthermore, data items should be distributed as evenly as possible among all the nodes forming the network, avoiding clustering of information in sub-areas; even dissemination of data items should leverage lower access latency to any item from any node in the network. An even distribution of information in the network implies that whenever a data item is requested by a node S, the distance to the node that provides the reply is approximately the same, regardless of the location of S. Naturally, the actual distance depends on multiple parameters, such as the number of nodes in the network, the size of the cache where data items are stored, and the number of data items. From a latency point of view, one should aim at minimising distance (i.e., ideally, any data item should be available from one of the 1-hop neighbours of S). Finally, since in wireless ad-hoc networks both bandwidth and battery power are precious resources, the algorithm should also minimise the amount of signalling data.

Based on these goals, this paper introduces PCache, an algorithm for efficiently spreading and retrieving cached data items in wireless ad-hoc networks. The algorithm provides two separate operations: dissemination and retrieval of cached data items. The implementation of these operations is orchestrated such that a limited number of messages is required for retrieving any data item from the network, independently of the addition, removal or movement of nodes. This goal is achieved by a combination of four different complementary mechanisms: an efficient best-effort probabilistic broadcast mechanism; a distributed algorithm for deciding which nodes replicate a given data item; a data shuffling mechanism to improve the distribution of data replicas and an expanded ring-search mechanism to support queries.

There are several interesting applications for our algorithm. For example, PCache can be used to implement a distributed name service (nodes would advertise their domain name and address), a service discovery protocol, or a

directory service for a peer-to-peer file system.

The rest of the paper is divided as follows. Section 2 describes the PCache algorithm in detail, illustrating its working principles and parameters. Section 3 evaluates and comments the algorithm, by presenting the results of extensive simulations. Section 4 presents related work, and finally, Section 5 summarises the main issues raised in this paper and presents pointers for future work.

2 The PCache Algorithm

Each node in a PCache system has a cache of a limited and predefined size. The cache is used to store a fraction of all the data items advertised. Each data item is composed of a key, a value, an expiration time and a version number with application dependent semantics. Nodes continuously pursue a better distribution of the items, by varying the content of their caches. The goal of PCache is to provide an adequate distribution of data items so that each node is able to find a significant proportion of the total items in its cache or in the cache of the neighbours within its transmission range.

PCache is a reactive protocol in the sense that it only generates packets to satisfy the requests of applications. PCache provides two distinct operations: data dissemination and data retrieval. These operations are implemented using three types of messages. In the dissemination process, nodes cooperate to provide an adequate distribution of the replicas of new or updated versions of data items. *Dissemination messages* are broadcast following an algorithm to be described later. The retrieval process is triggered by applications requesting to PCache the value associated with a key. The protocol first verifies if the value is stored in its local cache and if it is not, it broadcasts *query messages*. Nodes having in their cache the corresponding value address a *reply message* to the source of the query.

This section begins by presenting the structure of the cache at each node and the content of PCache messages. It then provides a description of the dissemination and retrieval processes and of the handling of the Complementary items.

2.1 Cache Structure

Nodes in a PCache system provide storage space for caching a fraction of the items advertised by all nodes. Nodes always try to keep their caches full, occupying all free space before beginning to overwrite other cache entries. The system does not require the caches at all nodes to be of the same size but proposes a common format and cache update policy.

Each data item is stored in cache together with auxiliary information to support an adequate distribution of replicas (see Figure 1) . The popularity ranking counts the number of times that a node listened for the item in the messages received previously. The *eraseCandidate* flag helps to leverage item distribution by suggesting the items that are more adequate for replacement.

A data item is said to be owned by the node that initiated its advertisement after an application request. To ensure that at least one copy of each item exists, nodes do not replace the items they own with items advertised by other nodes. It is assumed that owned items are stored in a separate region of the

```
cached item {
    type: {owned,remote}
    key: opaque
    value: opaque
    expiration: time
    version: int
    popularity: int
    eraseCandidate: bool
}
```

Figure 1: Structure of PCache items in cache

```
type: {dissemination,query,reply}
time to live: int
source: address
serial number: int
time from storage: {0,1,2}
route stack: address[]
# items
items: [] {
    key: opaque
    value: opaque
    expiration: time
    version: int
}
```

Figure 2: Content of a PCache message

memory space of the devices so that the space available for caching third-party records is kept constant.

2.2 Message Content

PCache messages share a common header that describes the type of message (dissemination, query or reply), a *time to live* (TTL) field, decremented by each node that forwards the message, and additional information concerning the items it carries and their relation with the state of the cache of other nodes. The content of PCache messages is presented in Figure 2.

The fields *source*, containing the address of the node that created the message and *serial number*, containing a number local to each node and incremented at every message it creates, are used to uniquely identify a PCache message. To identify duplicates, nodes keep a record of the messages recently received. In PCache, it is common to have messages to be forwarded and changed by multiple hops. We define the source of a message as the node who created it and defined the value for the *source* and *serial number* fields. Messages are edited and forwarded by multiple *senders* which are not allowed to change the content of

these fields.

Items are stored in an application dependent format, transparent for PCache. Similarly to some routing protocols ([10]), query messages accumulate the path to be used by a reply in a field, here identified as *route stack*. The header also carries information to help to leverage the distribution of items. This is the case of a field named *time from storage* (TFS).

2.3 Broadcast Algorithm

A broadcast algorithm is used for forwarding dissemination and query messages, although with some differences, highlighted in the respective sections. It should be noted that the algorithm does not intend to deliver the messages to all nodes with high probability. Instead, the retrieval of a data item from the network is guaranteed by the combination of both the dissemination and retrieval procedures and by the replication of the data items. This allows to use an unreliable broadcast algorithm, focused on the reduction of the number of messages. The broadcast algorithm puts together a mechanism to limit the number of retransmissions in floodings (similar to those in [7, 14]) and a protocol that uses the receiving power of messages to optimise the propagation of the flooding [11, 14] and adapts them to an environment where the expectations of delivery are lower. It is assumed that the reception power of a message can be provided by the network card driver.

The algorithm tries to reach the biggest number of nodes with the lowest number of transmissions. Therefore, the algorithm privileges retransmissions performed by nodes located farther away from the previous sender, which have a higher probability of reaching a bigger number of the nodes that have not yet received the message. To limit the resource consumption of the nodes, the algorithm also prevents from retransmitting nodes whose contribution to the number of nodes covered is believed to be small.

For each message m, the broadcast algorithm works as follows. Each node receiving for the first time a copy of m will place it on hold. The hold time is proportional to the power with which the message was received. Disregarding any fading effects in the wireless media, it is expected that nodes more distant to the sender of m have a lower holding period. During the hold period, each node counts the number of duplicates of m it receives. Preliminary simulations showed that in the majority of cases, a node listening to at least two retransmissions can discard the message without negatively influencing the message dissemination expectations. Therefore, a node will retransmit m if it has listened to less than two retransmissions of the message. The message will be marked for dropping otherwise. The handling of a message is dependent of its type and further described in the following subsections.

2.4 Dissemination Process

The rationale for the dissemination process is better explained assuming a configuration where nodes do not move, and that nodes have unlimited cache size (so that entries in the cache are never replaced). In this scenario, the dissemination process provides a reasonable probability that all items are found within the transmission range of every node. However, even in situations of limited cache size and nodes mobility, the algorithm provides a reasonably even distribution

of data items, as Section 3 will show. The dissemination algorithm mandates that, starting at the last storage, every third node propagating a dissemination message stores the advertised items. Complete determinism is removed from the algorithm by permitting other intermediary nodes to store the record, although with a small probability. In principle, this approach allows any intermediary node to have a copy within its 1-hop neighbourhood; the copy would be located either in the node from which a message was received, or in the next hop (if it exists).

Dissemination of data items is triggered by the source node with the broadcast of a dissemination message. In dissemination messages, the *time from storage* (TFS) field indicates the distance (in number of hops) from the sender to the closest node that is known to have stored the items. Therefore, the source node sets the TFS field to zero to indicate that the records are stored in its local cache.

Each node receiving a dissemination message places it on hold for a period of time proportional to the reception power, as described in Section 2.3. During the hold period, the node counts the number of retransmissions listened and calculates mintfs, which is the lowest value of the TFS from the original message and of all retransmissions. At the end of the hold period, mintfs will indicate the distance in hops to the closest node(s) that stored a copy of the item.

When the hold period expires, the node uses the number of retransmissions listened, mintfs and a random number generator to decide for one of three possible actions:

- If a node listens to two or more retransmissions and mintfs is 0 or 1, following the rationale of the broadcast algorithm, it can safely discard the message. Listening to two or more retransmissions suggests that the propagation of the message in the neighbourhood is being assured through some of the neighbours, so there is no need to further forward the message. A low value of mintfs (0 or 1) indicates that a close neighbour has stored the message, so it is advisable to reserve space in the cache for items carried in another message.

- The data item is stored in the node's cache and the message is retransmitted. This will be the action to execute with probability $e^{mintfs-2}$ if the first criteria did not apply. The probability of storing an item increases with the distance to the closest copy. In particular, if the closest copy is three hops away (signalled by a mintfs of two) a copy will always be stored in the node. Nodes executing this alternative will forward the dissemination message with the TFS field set to 0. As a consequence, the mintfs of neighbour nodes that did not terminated the hold period will be set to the lowest possible value and will have their probability of storing the item reduced. From the above, it can also be concluded that TFS and mintfs are always bound between 0 and 2. PCache benefits from having some randomisation associated with the decision of storing an item. $e^{mintfs-2}$ has shown to be adequate because it presents an exponential grow with the distance to the closest copy. The probability of storage for nodes with mintfs of zero or one is respectively 0.14 and 0.36.

- A message will be forwarded but the data will not be stored in the cache if none of the previous conditions applied. The TFS of the retransmission

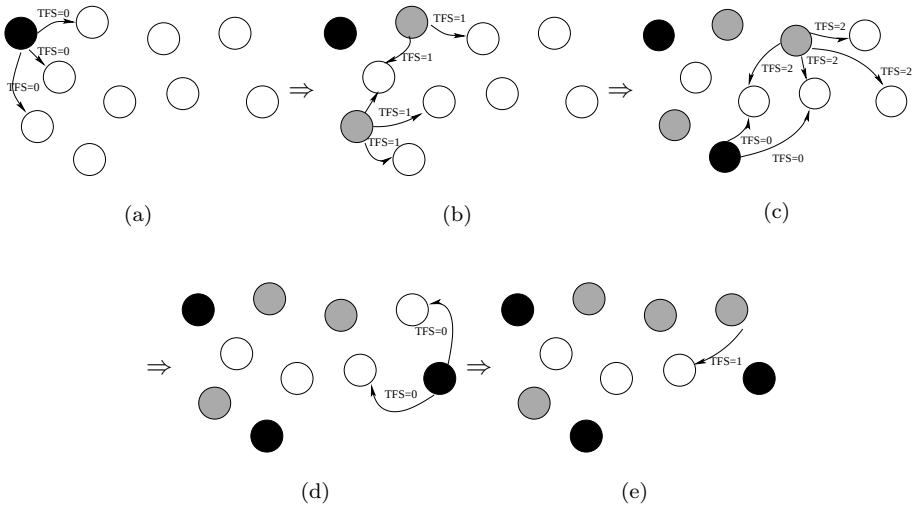


Figure 3: Progress in dissemination of an item

will be set to mintfs+1 to inform the listening nodes of the additional hop to the closest node that stored the item.

Figure 3 exemplifies a dissemination where all nodes are able to retrieve the item in its 1-hop neighbourhood. Nodes that forwarded the message are represented in gray and nodes that stored and forwarded the item in black. Three copies of the item were stored. The first at the source node (Figure 3(a)), the second for probability (Figure 3(c)) and the third because the node had a mintfs of two (Figure 3(d)). For clarity, only a subset of the message receptions are represented.

2.4.1 Analytical Evaluation of the Number of Copies Stored

To make the analysis of the number of copies stored by the dissemination algorithm manageable, it was necessary to impose some constraints to the modelled environment. It is assumed that each execution of the dissemination algorithm progresses in rounds. That is, that each node considering to transmit in round r has received the original transmission and any retransmissions in round $r-1$ and that all nodes transmitting at round r can not influence the decision to transmit of any other node that has also received the transmission in round $r-1$. The analysis will consider only the nodes that decide to retransmit since the remaining do not actively contribute for the outcome and do not influence the decision of nodes that retransmit.

The number of copies of each data item stored in the network during a dissemination phase in PCache is estimated using three interdependent functions.

$T(r,t)$ for some message received by a node in round r, returns the probability of the message having TFS t.

$m_T(r,t)$ gives the probability of mintfs being t for a node in round r that at the end of the holding period has decided to retransmit;

$S(r)$ returns the probability of a node in round r to have stored the item;

Consider a node in round r that received a message m transmitted by node s in round $r-1$. The TFS of the message will be 0 if s stored the item and 1 or 2 otherwise.

$$T(r,t) = \begin{cases} S(r-1) + (1-S(r-1))m_T(r-1,2), & t = 0 \\ (1-S(r-1))m_T(r-1,t-1), & 1 \leq t \leq 2 \end{cases} \quad (1)$$

Node s may have stored the item for two reasons, each represented by a parcel of the sum in the first row of Eq. 1. The first parcel of the sum gives the probability that the item had been stored as a result of the randomness of the algorithm and the second the case where the item was stored because mintfs was 2. The second row considers the probabilities of the previous node not having stored the item. Algorithm rules dictate that the value advertised in the TFS field should be one above the mintfs determined by s. Therefore, the probability of receiving m with TFS t is the same as the probability of s did not stored m and had a mintfs of $t-1$.

The mintfs value of each node depends on the TFS of the transmissions of m it receives. All the possible situations are arranged in four cases, relevant for the exposition:

1. The node received only the original transmission. In this case, mintfs will be the value of this message;
2. The node received the original transmission and one retransmission. In this case, mintfs will be the lowest TFS of both messages;
3. The node received any number of messages, all having the TFS field set to two. For the purposes of this analysis, this can be considered a particular case of the previous one, when the first two messages received had TFS= 2;
4. The node received at least three messages, the original and two retransmissions and at least one of them had TFS lower than 2. In these conditions, the node will not store or retransmit the data item, therefore, this case is not relevant for the analysis.

Function m_T is defined considering the first two cases. Since the number of messages depends on the network topology, and can change, for the same topology for each source node, it is assumed an equal probability of the holding period to expire after the node had received one or two messages. m_T is defined as:

$$m_T(r,t) = \begin{cases} 1, & r = 1 \wedge t = 0 \\ 0, & r = 1 \wedge t \geq 1 \\ \frac{T(r,0)}{2} + \frac{T(r,0)(T(r,0)+2T(r,1)+2T(r,2))}{2}, & r > 1 \wedge t = 0 \\ \frac{T(r,1)}{2} + \frac{T(r,1)(T(r,1)+2T(r,2))}{2}, & r > 1 \wedge t = 1 \\ \frac{T(r,2)}{2} + \frac{T(r,2)T(r,2)}{2}, & r > 1 \wedge t = 2 \end{cases} \quad (2)$$

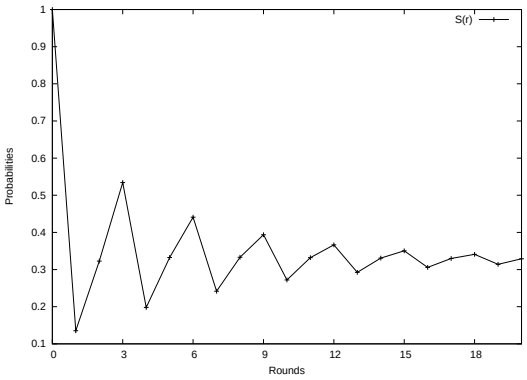


Figure 4: Probability of storage for nodes located at different rounds

mintfs is only evaluated for rounds after the initial dissemination. Therefore, the function is not defined for $r = 0$. The results of the function for the first round are immediate: in round 0 only one message is transmitted, with TFS= 0. Therefore, mintfs for nodes in the first round must be zero. Other values of mintfs will occur with probability zero.

For rounds other than the first, m_T evaluates the probability of receiving all the permutations of 1 or 2 messages and arranges them accordingly to the mintfs of each permutation. We remind that the probability of receiving each individual message is given by $T(r, t)$. Since it is assumed that retransmissions in the same round are independent, the probability of receiving some pair of messages is given by the product of the probability of receiving each of them. In Eq. 2, for each t, the first component of the sum accounts the probability of receiving only one message, with TFS= t. The second component considers all possible combinations of pairs of messages, for which the minimal TFS will be t. Each of the components is divided by two to distribute equally the probabilities between both cases.

With the auxiliary functions defined, it is now possible to proceed to the definition of $S(r)$ which will return the probability of a node that forwarded a dissemination message in round r had also stored the item in its local cache. $S(r)$ is presented in Eq. 3

$$S(r) = \begin{cases} 1, & n = 0 \\ \sum_{t=0}^{2} e^{t-2}.m_T(r,t), & n > 0 \end{cases} \quad (3)$$

$S(0)$ is 1 to comply with the requirement that the source node always stores the items it advertises. For nodes in other rounds, the probability will be given by weighting the probability of the occurrence of each mintfs with the exponential function introduced in the algorithm description.

Figure 4 applies $S(r)$ to the first 20 rounds. As expected, it can be seen that the algorithm creates a local maximum at every third round followed by a local minimum. As the message progresses for more distant rounds, so the differences of the probabilities become more attenuated, to attend to the wider region covered at each round.

This section has shown that the algorithm adapts the storage probability

to the distance to the source of the item. The adequacy of this algorithm for retrieving items will be experimentally evaluated in the following section. To conclude the analysis we notice that the number of replicas expected to be stored at each round could be derived by combining $S(n)$ with the number of retransmissions on that round. This is deferred for Section 3.2 where the number of messages collected from a simulation will be used to estimate the number of cached copies of data items.

2.5 Retrieval Process

To retrieve some value from the network, a node begins by looking for the key in its local cache. If the value is not found, the node prepares a query message, placing the key in the message. An expanding ring search is performed due to the expectations that the dissemination process was able to store the value in the 1-hop neighbourhood of the node. The message is first broadcast with TTL equal to one. The query is reissued with a large TTL if no reply is received within some predefined time limit. The protocol imposes a limit on the number of retries to be performed, which occur at growing time intervals.

A node receiving a query message, and that does not find the value in its local cache executes the broadcast algorithm described previously. The message is retransmitted if TTL permits and the node does not listen to the broadcast of two copies of the original query. Prior to broadcasting, the sender appends the address of the previous hop to the *route stack* field, in a process similar to the route discovery algorithm in some source routing protocols for MANETs, like DSR [10].

If a node receiving a query message finds the requested key in its cache, it does not enter the holding period of the broadcast algorithm. Instead, it sends a point to point reply to the source of the query. The reply message is sent after a random delay, to prevent the collision of multiple replies. PCache makes no provision to limit the number of replies addressed to a node. Therefore, one reply from each node listening to the query and with a replica of the item in its cache can be received. The expanding ring search is expected to limit the number of replies in the majority of the cases. When the probability of finding an item in the 1-hop neighbourhood is low, PCache can be adapted to reduce the number of query floods and therefore, reduce the number of replies, by slightly changing the expanding ring search procedure. The TTL for the queries can start at an higher value (e.g. 2) to cover a larger number of nodes in the first round or can increase progressively until a reply is found. A comparison between TTL initial values of 1 and 2 is presented in Section 3.5.

The path constructed in the query message identifies a sequence of hops that were unable to provide the requested value. The reply message follows this path, with each intermediary hop addressing it to the one that preceded it in the query propagation. The item is stored at the node that issued the query. It should be noted that PCache is orthogonal to the underlying routing protocol. If no ad-hoc routing protocol is available, PCache unreliably sends the message to the network without verifying if the destination is still in range. The use of a routing protocol for MANETs for delivering reply messages would increase the network traffic as a flooding could be required to find a route between the replying node and the source of the query. On the other hand, PCache makes no provision to limit the number of replies sent to the node. Therefore, there is

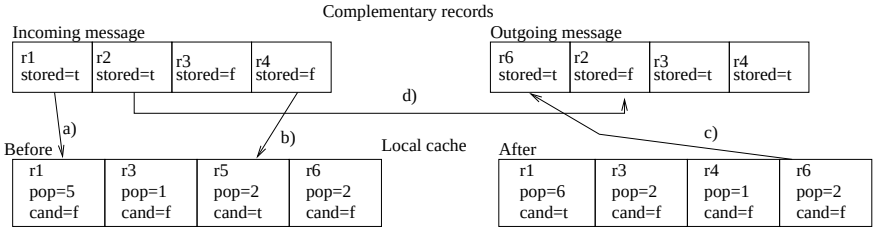


Figure 5: Operations performed over Complementary items when forwarding a message.

a reasonable probability that at least one of the routes constructed during the query propagation remains valid until the reply is delivered. It should be noted that this assumption is similar to that of many reactive routing protocols for MANETs [10, 13] for route discovery.

2.6 Additional Features

2.6.1 Complementary Items

In complement to the information relevant for the action in progress, in PCache, all messages carry as many *Complementary items* as possible, without exceeding a predefined maximum message size. The role of Complementary items is to leverage an even geographical distribution of the information by mixing fractions of the cache of different nodes that have been forwarding the message and by letting neighbour nodes learn relevant information about the state of each other caches, without requiring a membership protocol. These items are not relevant for the operation taking place, but help in propagating data throughout the network. Complementary items are handled similarly regardless of the type of the message that carries them.

The transmission of a message has a non-negligible fixed cost that is independent of the message size. In the IEEE 802.11 protocol family, for example, this cost can be attributed, among others, to the fixed size of the MAC and Network level headers and to the contention period. Transmitting the complementary items adds to the size of PCache messages sent over the wireless medium. However, previous analysis [3, 5, 9] show that bigger packet size in IEEE 802.11-based ad-hoc networks do not linearly increase the power consumption of the mobile devices or decrease the throughput. In PCache case, they improve the replication and geographical distribution of the data. We assume that applications advertise small sized items, so that at least a few are able to fit, together with the PCache and other headers, in a single frame at the link layer level. The behaviour of PCache using messages of different sizes and different number of items is evaluated in Section 3.

Each Complementary item carries a flag, *storedInSender*, which is active if the item is stored in the sender cache. When preparing the Complementary items in a new message, the source node places the items it owns and, if there is still space available, the less popular items of its cache (according to the popularity ranking). The *storedInSender* flag is active for all items to indicate that all Complementary items are present in the sender's cache.

The different operations over Complementary items are presented in Figure 5. All nodes receiving the message update the auxiliary information of the items simultaneously present in their caches and in the Complementary items of the message. These items have their popularity ranking increased. The *eraseCandidate* flag associated with the item in the node's cache becomes active if the item was presented in the message with its *storedInSender* flag on. The rationale is that if some neighbour node already stores a copy of the item in its cache, then a better distribution is achieved if the slot was occupied by some other item. Action a) of Figure 5, for example, sets the *eraseCandidate* flag of item r1 to true, since the item is already stored in at least one of the node's neighbours.

A node that decides not to drop a message (according to the criteria presented in previous sections) makes use of the Complementary items to change its local cache. Each Complementary item not present in the node cache and with the *storedInSender* flag off may be stored locally with probability p_{ins} if the cache has space available or any of its entries has the *eraseCandidate* flag active. The item preferably occupies a free slot in the cache. This was the case in action b), where r4 replaces r5, which is known to be stored by some of the node's neighbour.

All nodes forwarding a message can change the Complementary items it carries. Data items owned by a node will replace random positions of the Complementary items. An item simultaneously present in the Complementary items and in the local cache may be replaced in the outgoing packet by the sender with probability p_{rep}. The items to be inserted are those that have a lower popularity ranking in the node's local cache. Action c) shows the case were the forwarded message has the item r1 replaced by item r6, which has the lowest popularity ranking from those not present in the message. Finally, the node is required to update the *storedInSender* flags of all Complementary items in the message so that they reflect the state of its cache. See for example action d), which updates the flag to false, to reflect the fact that r2 is not kept in the node's local cache and also the remaining items, which have their flag set to true to indicate that the items are present in the local cache.

2.6.2 Data Update and Expiration

PCache provides two mechanisms for the update of cached data items. The *expiration* field defines a lifetime for the item. All nodes with the item in their cache will remove it when the predefined time expires. The *version* field provides a mean for updating cached information. Updating a cached data item with a newest version is an operation that takes precedence over all others previously described. Before deciding if a message will be forwarded or replied, nodes compare the content of their caches with all the items in the message (including Complementary items) and unconditionally update the local cache if the message carries a record with a more recent version number than the one available in the local cache.

2.6.3 Robustness Considerations

A premise of ad-hoc networks is the permanent addition and (possibly temporary) disconnection of nodes. PCache is robust against frequent nodes depar-

tures or medium impairments. Because PCache does not rely on a membership protocol, the only impact of node removal is a reduction of the number of replicas of some of the items in the system. The addition of a node to the network also does not imply any message exchange. The node will begin to fill its cache as soon as dissemination and query messages start to be listened by the node.

3 Simulation Results

A prototype of PCache was implemented in the *ns-2* network simulator v 2.28. The simulated network is composed of 100 nodes uniformly disposed over a region with 1500mx500m. Nodes move accordingly to the random way-point model [10] using three different speed models: 0m/s, 3-7m/s and 5-15m/s. In the latter cases, pause times are randomly selected between 0 and 20s. 10 movement files were randomly generated for each speed. The simulated network is an IEEE 802.11 at 2Mb/s. Network interfaces have a range of 250m using the Free Space propagation model.

Runs are executed for 900s of simulated time. Each run consists of 100 disseminations and 400 queries. Each node disseminates one data item in a time instant selected uniformly from the time interval between 0 and 400s. Items are uniquely identified by numbers between 0 and 99. Simulations do not consider expiration of cache entries since they could unfairly improve the performance of the protocol by freeing additional resources on the caches of the nodes. Queries start at 200s and are uniformly distributed until the 890s of simulated time. The nodes performing the queries and the queried items are randomly selected. The simulation ensures that only advertised records can be queried so that the evaluation of the protocol does not become obfuscated by bogus queries. No warm up period is defined. 10 traffic files were generated.

The sensitive of PCache to different parameters is evaluated by testing the parameter with different values while keeping the remaining parameters consistent with the baseline configuration. In the baseline configuration, the cache of the nodes was defined for accepting at most 10 items, excluding owned items, which are stored in a separate region of the memory. Each data item has a size of 250 bytes (50 for the key and 200 for the value). In this configuration, a full cache occupies about 3KBytes, which is a small value. Bigger values of cache size improve the performance of PCache, as there is more storage space available in the network.

The message size was limited to 1300 bytes. After removing the space required for the PCache header (estimated to be 13 bytes for the fixed part), a PCache message will carry at most 5 data items.

Unless stated otherwise, all values presented below average 10 independent runs, combining one movement file and one traffic file. The performance of the protocol is evaluated using the following metrics:

Average distance of the replies (DR) Averages the distance, in number of hops, from the querying node to the source of the first reply received. The average distance of a reply will be 0 if the value is found in the cache of the querying node.

Average nodes without an item in 1-hop neighborhood (N1) At the end of the simulation, and for each data item d, accounts the number of nodes

p_{ins}/p_{rep}	0.0	0.2	0.4	0.6	0.8	1.0
0.0	0.97	0.97	0.98	0.98	0.98	0.98
0.2	0.97	0.96	0.98	0.98	0.98	0.99
0.4	0.97	0.98	0.98	0.98	0.99	0.98
0.6	0.98	0.99	0.98	0.98	0.99	0.99
0.8	0.97	0.98	0.97	0.98	1.00	0.99
1.0	0.97	0.97	0.98	0.99	0.99	1.00

Table 1: Average distance of the replies for speed 0ms^{-1}

p_{ins}/p_{rep}	0.0	0.2	0.4	0.6	0.8	1.0
0.0	0.97	0.96	0.96	0.97	0.98	0.99
0.2	0.95	0.96	0.96	0.96	0.97	0.97
0.4	0.97	0.96	0.96	0.97	0.96	0.98
0.6	0.96	0.98	0.96	0.96	0.98	0.97
0.8	0.97	0.95	0.96	0.98	0.96	0.97
1.0	0.97	0.97	0.96	0.98	0.98	0.99

Table 2: Average distance of the replies for speeds 3-7ms^{-1}

that do not have a replica of d in its 1-hop neighborhood. The value is the average of this count for the 100 data items. This metric was only evaluated for scenarios with speed 0.

Number of dissemination/query/reply messages and bytes This metric measures the number of messages/bytes sent to the network for each PCache type of operation. Each forwarding by a node is accounted as one message and contributes with the size of the message (at the MAC level) to the total number of bytes.

3.1 Sensitivity to Probabilities

Tables 1 to 4 present the DR and N1 metrics for different combinations of p_{ins} and p_{rep}. These constants, which have been introduced in Section 2.6, dictate respectively the probability of having a Complementary item to be inserted in a node's cache and the probability of replacing a Complementary item before forwarding a message.

All values for the DR metric are in the interval $[0.95, 1.0]$. Because the distance between the extremes of the interval is of 5%, it can also be said that PCache is not particularly sensitive to the values of these probabilities. The tables show that for some combinations of p_{ins} and p_{rep}, PCache performs better when nodes move. This effect is attributed to the tendency of the random way-point model to concentrate nodes at the center of the simulation space [1]. Since node density increases at the center, so does the probabilities of having at least one of the neighbors with the values requested in cache. For this reason, the evaluation of experimental results will privilege the case where nodes do not move.

The best values for the DR metric are not coincident with those for the N1 metric. It should be noted that the results provided by DR depend of the random pattern of queries and directly affect the distribution of the items,

p_{ins}/p_{rep}	0.0	0.2	0.4	0.6	0.8	1.0
0.0	0.98	0.97	0.96	0.98	0.98	0.98
0.2	0.96	0.95	0.96	0.96	0.97	0.97
0.4	0.98	0.95	0.95	0.98	0.96	0.97
0.6	0.97	0.97	0.97	0.97	0.97	0.98
0.8	0.96	0.97	0.96	0.97	0.97	0.98
1.0	0.97	0.98	0.96	0.96	0.98	0.98

Table 3: Average distance of the replies for speeds 5-15ms^{-1}

p_{ins}/p_{rep}	0.0	0.2	0.4	0.6	0.8	1.0
0.0	8.50	8.69	8.78	8.74	8.22	8.68
0.2	8.97	9.26	9.35	9.18	9.21	9.21
0.4	9.42	8.81	9.37	9.48	9.31	9.78
0.6	9.54	9.53	9.06	9.39	9.28	9.96
0.8	9.95	9.50	9.64	9.79	9.60	9.92
1.0	9.95	9.33	9.83	10.09	9.53	10.17

Table 4: Metric N1 for different combinations of probabilities

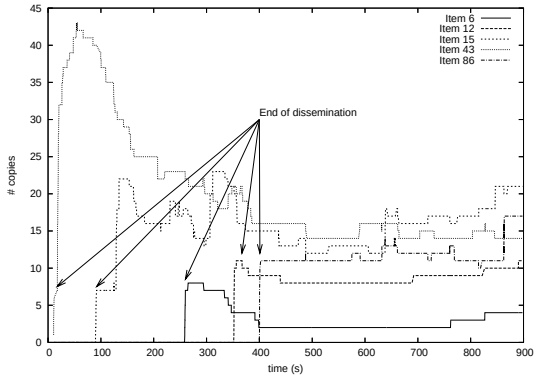
specially because a querying node will store the reply in its local cache. On the other hand, N1 is a *post-mortem* analysis of the distribution of the items, which has already been biased by the queries performed to measure N1.

The results suggest that low probabilities are preferable. Contrary to the intuition, the use of high probabilities does not increase the heterogeneity of the caches. Instead, it prevents nodes from disseminating their records beyond their 1-hop neighbors. The *storedInSender* flag prevents the receivers of a message from storing the records inserted in the previous hop but an high value for p_{rep} mandates the nodes to replace them before retransmission, restarting the loop. The possibility of not including Complementary records in the messages has different undesirable effects, that should be prevented. It slows the speed at which joining nodes will fill their cache, therefore worsening the performance of the protocol. Furthermore, even when not inserted or replaced in messages (as is the case when $p_{ins} = p_{rep} = 0.0$), Complementary records are used by the 1-hop neighbors of the sender to learn which records are redundant, and making them candidates for replacement.

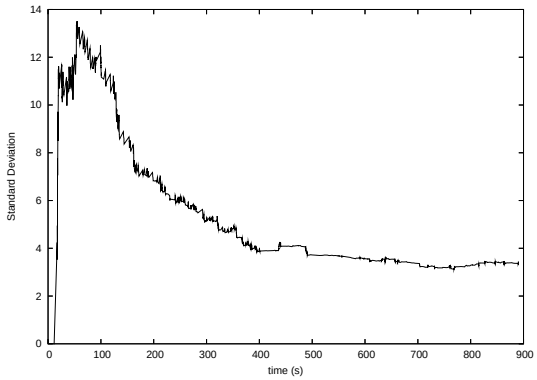
Values of $p_{ins} = 0.4$ and $p_{rep} = 0.2$ present the most consistent combination of results from both metrics and for all speeds. These values were selected for further analysis and are those used in all tests presented below.

3.2 Evolution of the Number of Copies

If the location of the nodes was ignored, the desirable distribution of items would divide equally the total cache space (given by the sum of the cache space of all nodes) by the number of items. For the baseline configuration presented above, and considering also the copy stored at the source node, each item should have 11 copies. Figure 6 show the evolution of the number of copies of some items and of the standard deviation of the number of copies of all items in one representative run of the tests above, with speed 0m/s, $p_{ins} = 0.4$ and $p_{rep} = 0.2$. In this particular run, the metric DR evaluated to 0.98 and the N1



(a)



(b)

Figure 6: Evolution of the number of copies and standard deviation in one simulated run of PCache

metric to 7.85. The pattern exhibited in these plots is similar to that observed in the remaining tests.

The data items represented in Figure 6(a) are those with the lowest (item 6) and highest (item 15) number of copies at the end of the simulation, of the first (item 43) and last (item 86) items disseminated and of one of the items that reached the end of the simulation with the average number of 11 replicas (item 12). All of them exhibit a common pattern which can also be found in a large majority of the items in all runs. In all of these cases, the number of replicas rises quickly from 0 to some value usually close to 8. From the inspection of the trace files, it was concluded that this fast climb is due to the dissemination phase. Later variations are a result of changes in the caches triggered either by Complementary items or the storage of records in querying nodes.

Table 5 applies the analytical model defined for PCache in Section 2.4 to

Round	S(r)	# Transm.	# Copies
0	1.000000	1.00	1.00
1	0.135335	5.52	0.75
2	0.322802	5.51	1.78
3	0.534140	4.33	2.31
4	0.198021	3.55	0.70
5	0.332449	2.77	0.92
6	0.441114	2.01	0.89
7	0.241633	0.90	0.22
8	0.333138	0.31	0.1
9	0.393713	0.05	0.02
Total	-	25.94	8.68

Table 5: Average number of messages per round

the run. The number of forwards on each round presented in the third column was extracted from the trace file of the simulation. The last column applies the function $S(r)$ to the values observed for the number of retransmissions. As it can be seen, the total number of replicas expected to be stored is compatible with the experimental results, for example with those presented in Figure 6(a).

The incorporation of the item in the Complementary items section of other dissemination messages justifies the large number of copies attained by the first item advertised in the simulation (item 43, in this run) and consequently, of the rapid increase in the standard deviation. Because the cache at the nodes is not filled, nodes forwarding the dissemination message of other items store the complementary items independently of p_{ins} (although the constrain of the storedInSender flag still holds). However, it can be seen that as the number of advertised items grows, the number of copies of the first item decreases, suggesting that PCache is able to balance the number of copies of each data item. The plot of the standard deviation of the number of replicas of the items, shown in Figure 6(b), supports this claim. After an initial grow it tends to stabilize around a small value (approximately 3.5) as soon as registrations finish.

Figure 7 is an histogram of the number of copies of each data item at the end of the simulation. For each data item, the graph presents the number of copies in excess (positive) or missing (negative) to the average number of copies. The distribution seems to be balanced, with more than 50% of the nodes in the interval $[-2, +2]$ and not presenting a significant number of peaks either for the positive or negative side.

Figure 8 shows in black the geographical distribution of the item with the minimum number of copies at the end of its dissemination phase. It can be seen that the algorithm was not able to put a copy near every node in the network. However, it was able to spread copies of the item over geographically distinct points of the simulated region.

3.3 Impact of Item Size

The impact of the size of the data item on the average distance of the replies is presented in Figure 9(a). In this set of experiments, the size of both the

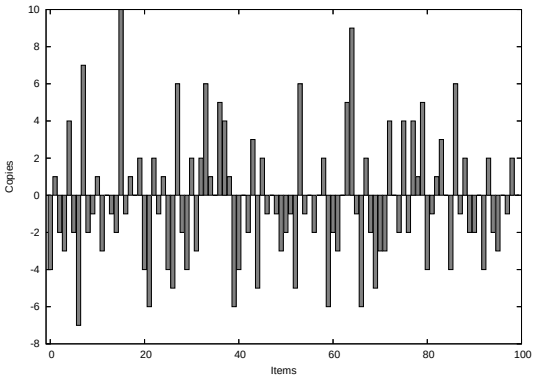


Figure 7: Difference to the average number of copies

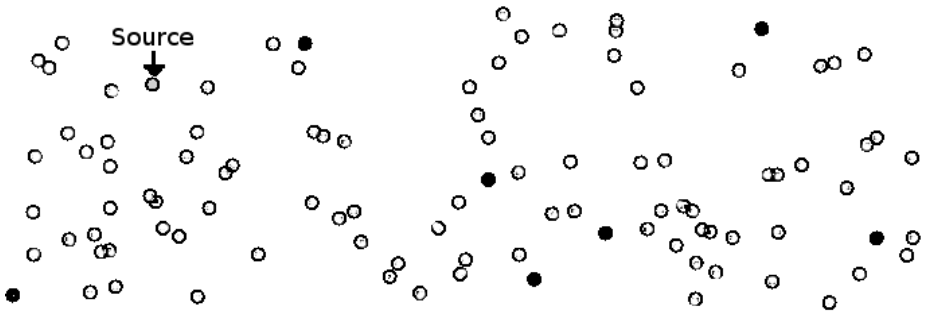
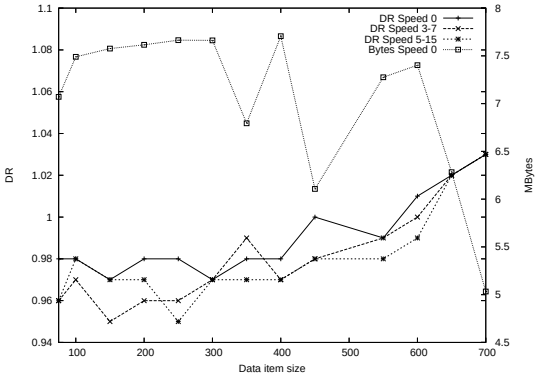


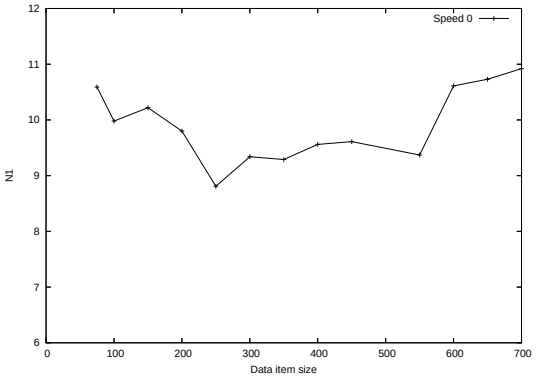
Figure 8: Dissemination of the item with the lowest number of copies at the end of the simulation

key and value components of the data item was increased while keeping the maximum size of the message constant. Results for the DR metric show that the performance of PCache degrades as the size of the data items increase. The total number of bytes transmitted by all nodes in the simulations for speed 0 is represented in the right y axis to justify that the performance degradation can not be attributed to the increased bandwidth consumption. (The total number of bytes transmitted in other speeds follows a similar pattern but is slightly lower. It was omitted for clarity.) Instead, we justify this behavior with the gradual decrease of the number of complementary items that can be inserted in messages. For values higher than 600 bytes and messages with maximum size of 1300 bytes, complementary records can only be carried in query messages, because only the key of the item to be retrieved is sent.

An excessive number of items in a message may also prevent complementary items from performing their role on leveraging the item distribution. As shown by metric N1 in Figure 9(b). For a large number of data items (i.e. when the data item occupies a small number of bytes), the absolute number of records that are inserted in cache and replaced in forwarded messages (dictated by probabilities p_{ins} and p_{rep}) increase and begins to adversely affect the cache distribution.



(a)



(b)

Figure 9: Variation of metric DR and N1 with the size of the data items

3.4 Sensitivity to Cache Size Ratio

To reduce query traffic, PCache aims at storing a significant proportion of the data items in the 1-hop neighborhood of every node. The number of cache entries available in the nodes located in the 1-hop neighborhood is conditioned by the size of the cache of each node and the number of neighbors.

Relevant for this analysis is the ratio between the size of the cache in the 1-hop neighborhood and the total number of items. This ratio, hereafter named Relative Neighborhood Cache Size ($RNCS$), is given by $RNCS = \frac{\sum_{i \in \text{neigh}} CS_i}{\#items}$ where CS_i is the cache size of node i in a neighborhood and $\#\ items$ is the total number of advertised items in cache. Assuming that the number of 1-hop neighbors is n and that all nodes have an equal cache size, the equation can be simplified to $RNCS = n\frac{CS}{\#items}$. The effect of the variation of each of these parameters in PCache was evaluated individually. In each of

the three tests, the remaining parameters were kept according to the baseline configuration described in the beginning of the section. Like in the rest of the paper, all results presented in this section are the average of 10 runs for each condition described.

Figure 10 labels the variations of $RNCS$ by changing the cache size of each node and of the total number of items respectively as "Cache Size" and "Items". For the variation of the cache size, the results were computed using sizes between 3 and 17 items at intervals of two. The number of items varied in the interval between 50 and 400 at intervals of 50.

To vary the number of neighbors, the baseline configuration was tested with the nodes configured for transmitting with different ranges while keeping the size of the simulated space constant. The tests were performed for transmission ranges between 150 and 325 meters at intervals of 25 meters.[1] The number of neighbors was estimated by averaging the number of nodes that received every broadcast message on every simulation with the same transmission range. The figure identifies the results of these tests with the label "Neighbors".

The resulting DR metrics are presented in Figure 10(a). As expected, performance of PCache increases with RNCS. For values of RNCS close to one, replies to queries will be found on average at 1.3 hops and at values close to 1.8 hops as the RNCS approaches 0.5. It should be noted that the values for RNCS equal to 0.5 where obtained by changing the number of data items to 400. Because the remaining parameters are always kept according to the baseline configuration, each individual node was able to store 3.5% of the total number of items.

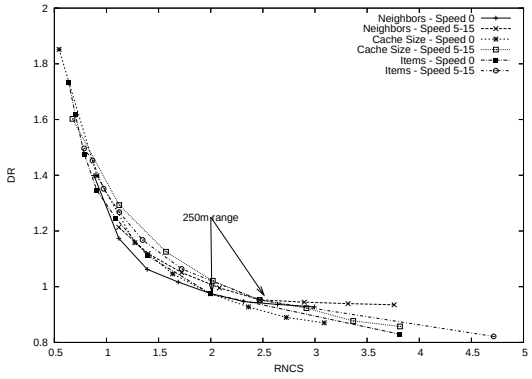
The figure reveals that the RNCS values are not coincident for the different speeds, even when tested with the same parameters. Again, we attribute this to the tendency of the random way-point movement model to concentrate nodes at the center of the simulated space [1], thus increasing the number of neighbors. The RNCS value corresponding to the baseline configuration at both speeds are highlighted in the figure. We emphasize also that a comparison between similar tests with different speeds is not immediate because for the same resources, different speeds result in different values of RNCS.

Because one of the criteria used to change RNCS makes the number of neighbors vary, the results for the N1 metric can not be compared. Therefore, Figure 10(b) presents N1 after an harmonization which consisted in dividing the results by the average number of neighbors. The results are consistent with those presented for DR. The figure shows that PCache can handle better a reduced number of neighbors but the benefit of adding new nodes has a limit when the total capacity of the neighborhood reaches twice the number of advertised items. On the contrary, with the increase of the size of the cache or the reduction of the number of items, PCache is capable of further improving its performance.

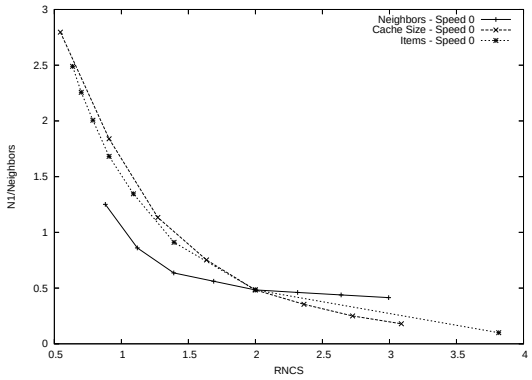
3.5 Number of Messages

Figure 11 shows the average number of transmissions per PCache operation in the tests of section 3.4 that changed the transmission range. The number of nodes in the network was kept unchanged. The number of messages per event account, for each type of message, the original transmission and all retransmissions performed during the execution of the broadcast algorithm or every

[1] Transmission ranges below 150m do not provide accurate results due to the large number of isolated nodes.



(a)



(b)

Figure 10: DR and N1 metric for variations of RNCS

forwarding of a point-to-point message. Although sharing a common broadcast algorithm, the figure shows a significantly smaller average number of messages of type query. This is due to the efficiency of PCache in item distribution. Part of the queries are replied locally and do not require a broadcast, while others may be replied by some of the 1-hop neighbors and therefore, require a single transmission. The decrease in the number of messages as the number of neighbors increases results from distinct factors for dissemination and query operations. In disseminations, it is due to the restriction imposed on the number of retransmissions. As the number of neighbors increases, so does the number of nodes who are capable to receive each other retransmissions and, therefore, do not retransmit. This factor also affects the flooding of query messages. However, as fig. 10(a) has shown, the probability of finding the queried item within the 1-hop neighborhood increases for higher density of nodes, thus reducing the number of floodings of queries performed. Finally, the figure shows that after

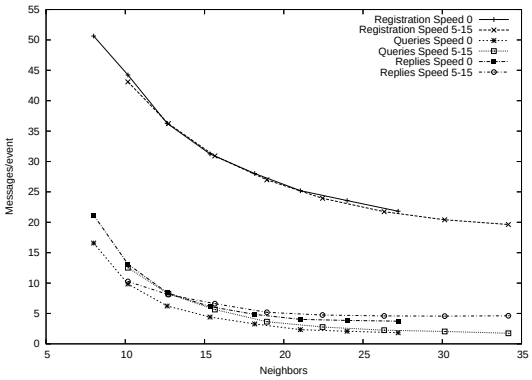


Figure 11: Number of messages per operation

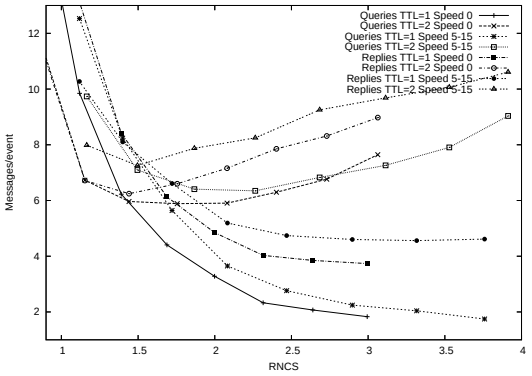


Figure 12: Comparison of the number of messages for queries and replies for TTL=1 and TTL=2

an initial decrease, the number of reply messages tend to stabilize, due to the excessive redundancy of replicas that leads to an increasing number of replies for each query.

The effect of two expanding ring search policies on the number of queries and reply messages is compared in Figure 12. For clarity, the results for the intermediate speed are omitted. The figure shows that performing the first step of an expanding ring search with TTL=2 is preferable when RNCS is low. These results were collected by changing the transmission range, in the tests described above. In this scenario, there is a low probability of having a query replied in the 1-hop neighborhood and, to prevent the subsequent flooding, it is more advantageous to perform an initial extended search, which has an higher probability of having the query replied. As RNCS increases, the number of queries replied with TTL=1 gain dominance. The propagation of the query to the second hop is useless in the majority of the cases and consumes additional resources both on the unnecessary propagation of the query and of redundant replies.

4 Related Work

So far, gossipping techniques in ad-hoc networks have been studied mainly for two applications. Gossipping can be used for spreading routing information among all the ad-hoc nodes, or can be utilised for distributing and replicating information. The algorithm we propose belongs to the second category, but we adapt some of the concepts presented in the routing gossipping algorithms.

A milestone ad-hoc gossipping-based routing paper is from Haas et al. [7]. The authors observe that gossipping is particularly affected by the gossipping probability and by the network topology. They find that a gossipping probability between 0.6 and 0.8 suffice for ensuring high reachability of nodes in the network and allows reducing the number of messages sent to the network by 35% compared to basic flooding.

One of the first papers addressing effective replica allocation in ad-hoc networks is from Takahiro Hara [8], where three algorithms are proposed for data replication. Such algorithms are not gossip-based, but rather, deterministic. The parameters taken into account for deciding, at every node, which replica to reallocate to another node or set of nodes depend on the access frequency to a given data item. Neighbourhood awareness is taken into account as well, by eliminating replica duplication among neighbouring nodes or group of nodes. The results show that neighbour awareness improves the accessibility of data item, at the expenses of more traffic in the network to maintain neighbourhood information. In comparison, the PCache algorithm is reactive and does not need any neighbourhood information, whose accuracy depends of external factors that can not be controlled.

Yin and Cao [15] propose an algorithm for cooperative caching in ad-hoc networks. Their model is slightly different from ours, as it assumes the presence of a single data source in the ad-hoc network; PCache is totally decentralised when considering the source of a data item. In [15], queries are unicast, sent to the data source, and if a node has either the data cached locally or the path to a node that holds the queried item, a reply is returned; otherwise, the request is forwarded to the data source. A hybrid approach combining the benefit of data and path caching is shown to be the best performing one. PCache can ensure faster query responses, and in a lower number of hops, due to its inherent broadcast nature, at the expense of having the querying node to receive possibly more than 1 reply for each issued query. In [15], only 1 reply is always returned for an issued query; this decreases the processing needed at the querying node, but increases the risk of never receiving a reply, due to the unstable nature of wireless ad-hoc networks. Another difference from PCache is that Yin and Cao assume locality of queries; the performance of the algorithm is strongly related to the degree of locality of queries. PCache, instead, assumes that queries are randomly executed from any node in the network to any disseminated item in the network.

Lim et al. [12] propose a novel caching scheme for Internet-based mobile ad-hoc networks. These network have access to the Internet, through one or more access points. Similarly to [15], there is a main source of data in the ad-hoc network, the access point. However, the querying scheme is broadcast; a four-way handshake is implemented to prevent nodes from receiving more than one reply to a given issued query. We deem that such a scheme is useful when the target data item is large in size, as would be the case of PCache used for

implementing a distributed file sharing system. If the advertised items are small, it may be faster to send directly the queried item. Note that also in [12], it is assumed query locality.

Autonomous gossipping [4] aims at finding an efficient way to spread and replicate cached information in the nodes of an ad-hoc network. In autonomous gossipping, the data items themselves try to identify other hosts which may be interested to the item, based on the data item's own profile and host's profile, advertised during registration phase. This approach is in contrast to the traditional push model were data items are injected in ad-hoc networks by the possessor nodes. Profiles are maintained in a distributed self-organising way, and updated using gossipping techniques. When data items arrive at a node, the autonomous gossipping algorithm is applied to decide what the data item will itself do, in an autonomous and self-organising fashion. Data items in a host decide whether continue to reside, migrate or replicate to another host. The difference with our approach is that we switch profiling for redundancy which may be more adequate for frequent connections and disconnections of nodes. The potential risk of our approach is that it may require additional storage space.

An interesting approach for spreading user data in sensor networks is described in [6]. The authors propose to enhance data availability in sensor networks by having sensors randomly distribute their cached contents to a random set of neighbours. If data received with a message do not fit into the small sensor node cache, then existing data are replaced by the new data. This shuffling ensures that an immediate neighbour gets a replica of the information being spread, and at the same time allows redistributing evenly the data among all the nodes of the networks. The major differences from PCache include the fact that the algorithm is proactive, as dissemination of items is done in periodic gossipping rounds. Further, the aim of the algorithm is to deliver cached data items to the sensor network sink node, and even dissemination comes as a side effect.

PCache has some similarities with peer-to-peer algorithms. All the nodes perform similar operations with respect to the algorithm, to increase its robustness and data availability. An algorithm for efficient file indexing in unstructured peer-to-peer networks has been proposed in [2]. The aim of the algorithm is of providing an efficient indexing scheme for searching files distributed in a peer-to-peer network; data structures called Bloom filters are used for the purpose. Filters are exchanged among nodes using gossipping mechanisms. The main difference from this approach and PCache is that it aims at retrieving files rather than smaller data items. Furthermore, PCache relies on gossipping mechanisms to spread the user data. Instead, the unstructured peer-to-peer method gossips the filters while the user data is queried and delivered deterministically.

5 Conclusions

This paper has presented PCache, an algorithm for retrieving and distributing cached information in ad-hoc networks. The algorithm is fully distributed as it does not assume the presence of only a few data sources in the ad-hoc network; each node, instead, can advertise own data items. The main goal of PCache is to ensure an even geographical distribution of the disseminated data items, so that requests for a given data item are satisfied by a pair of nodes as close as

possible with each other; ideally, by nodes that are in each other transmission range.

This goal is obtained by combining a mixture of different techniques, and a cooperative caching scheme. Broadcast messages are used to leverage the algorithm from the presence of an underlying routing protocol: we substitute the broadcast operations of route discovery typical of the most common routing protocols directly with PCache-level broadcasting. A probabilistic approach is taken to reduce the number of messages sent and to realize the even distribution of items in the nodes caches. Each PCache message carries additional information on the state of the cache of the sender allowing nodes to select data items that are most suitable to cache. This approach enables using neighborhood information without requiring nodes to run an expensive membership protocol. Simulations results show that PCache allows a fair dissemination of items throughout the ad-hoc network, and that in most of the cases, a node can find the item it requested already within the 1-hop neighborhood.

There are several interesting applications for PCache. A possible subject of future work can be to tailor a specific application for use with PCache. Possible candidates are name services applications, so that nodes advertise a combination of user name and contact address in the ad-hoc network. Nodes could query for a given user name (the key used in PCache queries) and retrieve the contact address as value for the requested key.

References

[1] C. Bettstetter, G. Resta, and P. Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(3):257–269, jul–sep 2003.

[2] A. Cheng and Y. Joung. Probabilistic file indexing and searching in unstructured peer-to-peer networks. *Elsevier Journal on Computer Networks*, June 2005.

[3] S. Ci and H. Sharif. An link adaptation scheme for improving throughput in the IEEE 802.11 wireless LAN. In *Proc. 27th Annual IEEE Conf. on Local Computer Networks (LCN 2002)*, pages 205–208, 2002.

[4] A. Datta, S. Quarteroni, and K. Aberer. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. In *Proc. International Conf. on Semantics of a Networked World(ICSNW 2004)*, pages 126–143, 2004.

[5] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. of the 20th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2001)*, volume 3, pages 1548–1557, 2001.

[6] D. Gavidia, S. Voulgaris, and M. Van Steen. Epidemic-style monitoring in large-scale wireless sensor networks. Technical report number: IR-CS-012.05, Vrije Universiteit, Amsterdam, the Netherlands, March 2005.

[7] Z.J. Haas, J.Y. Halpern, and L. Li. Gossip-based ad hoc routing. In *Proc. 21st Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2002)*, volume 3, pages 1707–1716, 2002.

[8] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proc. 20th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2001)*, volume 3, pages 1568–1576, 2001.

[9] T.-C. Hou, L.-F. Tsao, and H.-C. Liu. Analyzing the throughput of IEEE 802.11 DCF scheme with hidden nodes. In *Proc. of the IEEE 58th Vehicular Technology Conference (VTC 2003-Fall)*, volume 5, pages 2870– 2874, 2003.

[10] D. B. Johnson and D. A. Maltz. *Mobile Computing*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks, pages 153–181. Kluwer Academic Publishers, 1996.

[11] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of the 1st USENIX/ACM Symp. on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[12] S. Lim, W. Lee, G. Cao, and C. Das. A novel caching scheme for improving internet-based mobile ad hoc networks performance. *Elsevier Journal on Ad-Hoc Networks*, 4(2):225–239, March 2006.

[13] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[14] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless Networks*, 8(2/3):153–167, 2002.

[15] L. Yin and G. Cao. Supporting cooperative caching in ad hoc networks. *IEEE Transaction on Mobile Computing*, 37(1):77–89, Jan-Feb 2006.